Draft. Aimed at HSC.

PROVING TIGHT SECURITY FOR STANDARD RABIN-WILLIAMS SIGNATURES

DANIEL J. BERNSTEIN

Abstract. This paper discusses the security of the Rabin-Williams public-key signature system with a deterministic signing algorithm that computes "standard signatures." The paper proves that any generic attack on standard Rabin-Williams signatures can be mechanically converted into a factorization algorithm with comparable speed and approximately the same effectiveness. "Comparable" and "approximately" are explicitly quantified.

1. Introduction

In the Rabin-Williams signature system, (e, f, r, s) is a **signature** of a message m under a public key n if $e \in \{1, -1\}$, $f \in \{1, 2\}$, r is a B-bit string, and $fs^2 \equiv eH(0, r, m) \pmod{n}$. Here B is a fixed nonnegative integer (for example, 4), and H is a public function that maps strings to integers in an appropriate range.

This paper proves that any *generic* attack on the Rabin-Williams system—i.e., any signature-forging algorithm that works for all functions H—can be converted into an algorithm to factor n at comparable speed with probability $(2^B - 1)/2^{B+1}$, if the signer chooses the "standard signature" of each message. Standard signatures are defined in Section 3. "Comparable speed" is quantified in Section 5.

Of course, there might be a surprisingly fast factorization algorithm. There might also be surprisingly fast non-generic attacks on any particular function H. But the proof nevertheless provides some confidence that there are no silly mistakes in the design of the system.

Section 3 specifies all relevant details of the signature system. In particular, it presents a deterministic signing algorithm that produces standard signatures. Sections 4 and 5 present the proof. As background, Section 2 surveys similar signature systems, and explains why I'm focusing on this particular system.

Context. Fiat and Shamir in [10], after proposing another public-key signature system, proved that any generic attack against that system could be converted into a factorization algorithm with the "same" effectiveness and efficiency—although their notion of "same" was purely asymptotic and ignored polynomial factors.

Bellare and Rogaway in [3] published similar proofs for public-key encryption systems, and advocated designing systems to allow such proofs. I tend to avoid their "secure in the [uniform] random oracle model" terminology; "secure against generic attacks" is more comprehensible to the uninitiated.

Date: 2003.09.26. Permanent ID of this document: `c30057d690a8fb42af6a5172b5da9006`.

2000 *Mathematics Subject Classification.* Primary 94A60.

The author was supported by the National Science Foundation under grant CCR–9983950, and by the Alfred P. Sloan Foundation.

Proofs of this type were subsequently published for several modular-root ("RSA-Rabin-type") signature systems. There are three reasons that I'm not satisfied with the previous proofs:

- None of those systems are as efficient as the state-of-the-art system analyzed in this paper, namely the Rabin-Williams signature system with standard signatures. Section 2 discusses the relevant features of this system.
- Some of the proofs are merely outlined. I do not understand why this is tolerated, especially after the OAEP fiasco.
- The speed of the conversion is almost never stated (let alone optimized) in detail. For example, [5, Theorem 6.1] allows an added cost of "$[q_{\mathrm{sig}}(k) + q_{\mathrm{hash}}(k) + 1] \cdot k_0 \cdot \Theta(k^2)$"; this could be the dominant cost for large q's, but the reader has no way to tell without seeing an upper bound on Θ. Bellare and Rogaway suggest choosing key sizes in light of these theorems, but that requires quantification of effectiveness *and* efficiency. The theorems in [5] do not justify the 1024-bit/3447-bit example in [5, Section 1.4].

My goal in this paper is to produce a detailed, quantified, carefully optimized proof for the most efficient known signature system.

2. Why this particular system?

This section explains several useful features of the Rabin-Williams signature system with standard signatures.

This section can also be viewed as a comparison chart surveying the features of various modular-root signature systems. It is a pleasant surprise that *one* system—this system—manages to simultaneously provide *all* the features; I see no reason to use any modular-root system that does not provide all the features listed below. Of course, users are still faced with a difficult choice between modular-root systems (fast verification) and other systems (short keys and signatures).

Hashing. This system scrambles messages by feeding them to H: the signature of m is a root of a hash of m rather than a root of m itself. This is crucial for security.

History: The signature system proposed by Rivest, Shamir, and Adleman in [23] did not hash messages; it was trivially breakable. Rabin's system in [22] did hash messages; it remains unbroken today. The apparent security benefit of hashing was mentioned in [22, page 10, last sentence].

Reader beware: Many authors unjustifiably refer to an oversimplified, trivially breakable, non-hashing system as "Rabin's system"; consider, for example, the claim by Goldwasser, Micali, and Rivest in [12, Section 3] that "Rabin's signature scheme is totally breakable if the enemy uses a directed chosen-message attack." Furthermore, many authors—see, e.g., [24, Section 7.1]—describe hashing as merely a way to handle long messages, rather than as an essential component of the system no matter what the message length might be.

Small exponent. In this system, the exponent of s is a small fixed integer, rather than a large integer chosen randomly by the signer. This makes verification *much* faster. It also saves space in public keys.

History: The RSA system in [23] used large exponents. Rabin in [22, page 5] pointed out the huge speed advantage of small exponents. Reader beware: Many authors incorrectly credit the use of small exponents to [23] rather than to [22]. For example, Knuth in [17, pages 386–389] (and again in [18, pages 403–406]) explained an exponent-3 system and unjustifiably called it "RSA."

Exponent-3 proofs are just as easy as large-exponent proofs, but large exponents have inexplicably attracted more attention in this context. Bellare and Rogaway in [5] analyzed a traditional system that they called "FDH," and a system of their own design called "PSS," in both cases using large exponents. See [5, Section 2.1]; see also Coron's [6, Section 2.3] and [7, Definition 4]. Katz and Wang were exponent-agnostic in [13]: they stated their results for more general "claw-free permutation pairs," following [12] and a suggestion of Dodis and Reyzin. The "PRab" claim in [5, Theorem 6.1] used a small exponent, but the proof was merely outlined; [8, Theorem 4] used a small exponent, but no proof was given.

Exponent 2. This system uses exponent 2, as introduced by Rabin in [22], rather than exponent 3. This speeds up verification; it saves space when signatures are expanded for even faster verification; and, in the other direction, it allows more compression of signatures.

I've noticed that some programmers fear exponent 2: there appears to be a widespread belief that exponent-2 signing requires Euclid-type computations of Jacobi symbols. That belief is incorrect. See Algorithm 3.1.

Exponent 2 does require some extra effort in security proofs, because a uniform distribution of $s \bmod n$ does not correspond to a uniform distribution of $s^2 \bmod n$. There are several proof strategies here, depending on whether signers choose uniform random square roots, or square roots distinguished by being squares, or square roots distinguished by being absolute values of squares.

Message prefix: r. This system inserts a signer-selected B-bit string into each message. This was introduced in [22]: Rabin suggested $B = 60$, with a random choice of r.

It isn't clear whether r helps security, but it certainly helps these security *proofs*. Without r (equivalently, with $B = 0$), the proofs become quantitatively weaker as the signer signs more and more messages; see the "FDH" analyses in [5] and [6]. This problem disappears when B is sufficiently large.

Reader beware: Many authors have failed to give Rabin proper credit for his randomized signatures (r, s). For example, Goldwasser and Bellare have posted lecture notes (1) claiming that Rabin introduced a signature system with neither H nor r; (2) assigning credit to a 1983 paper of Goldwasser, Micali, and Yao for "pioneering" randomized signatures; and then (3) describing a "PSS0" system—randomized (r, s)—as if it were new. Similar comments apply to the "PFDH" system in [7] and [13].

Explicit transmission of r. This system includes r in signatures; the verifier does not need to reconstruct r. This is a prerequisite for the fastest signature-verification algorithms.

The "PSS"/"PRab" alternative, proposed by Bellare and Rogaway in [5], is to build a function H' from H so that r can be recovered from $H'(0, r, m)$, and thus from s, where $s^2 \equiv H'(0, r, m)$. This is incompatible with the fastest signature-verification algorithms.

No bad r's. This system has an s for every r. This speeds up and simplifies signing. It also allows deterministic r's and small B's, as discussed below.

History: The existence of s is trivial for injective powering. For squaring, the existence of s relies on the factors $e \in \{-1, 1\}$ and $f \in \{1, 2\}$ (and a related restriction on n) introduced by Williams in [25]. Rabin's signatures in [22] were, in retrospect, Rabin-Williams signatures with $e = f = 1$. With Rabin's system, s existed for only about 1/4 of all r's, so any choice of B below 8 would risk having users bump into some non-signable messages.

Deterministic signatures. This system chooses r deterministically (although not predictably!): viz., $r = H(1, z, m)$, where z is a secret held by the signer. Note that the same message will produce the same signature later. This idea was posted on `sci.crypt` by George Barwood in February 1997, and independently by John Wigley in March 1997.

This system then chooses e, f, s deterministically. See Section 3. This is standard practice: usually there are four possibilities for s in exponent 2, but implementations generate only one of the four. (Exception: Bellare and Rogaway in [5, Section 6] required randomization of s, which also means that r must be random and B must be large.)

The deterministic choice of signatures simplifies signing, because the signer does not need to generate a new random number for each message. It also allows small B's, as discussed below.

Tight security. Generic attacks on this system imply factorization algorithms with essentially the same success probability, even if many messages are signed.

This feature was introduced by Bellare and Rogaway with "PSS" in [5], and has appeared in most proofs since then. However, it has not been proven for the simplest ("FDH"; i.e., $B = 0$) systems, or for small B's in several other systems.

Short message prefix. In this system, B is small, although not 0; I suggest $B = 4$. This is useful because it makes r very short, saving space in signatures.

History: It was widely believed for years that a tight security proof required a considerably larger B. For example, I took $B = 256$ in a proof I posted several years ago, and $B = 128$ after seeing how Coron in [6] quantitatively improved the "FDH" analysis. But then Katz and Wang in [13, Theorem 2] pointed out that $B = 1$ allows a tight security proof *if* r's are chosen deterministically (but unpredictably). Thanks to Dan Boneh for pointing out [13] to me.

I'm suggesting $B = 4$ rather than $B = 1$ because it noticeably improves the success probability $(2^B - 1)/2^{B+1}$. I haven't seen any applications without room for the 3 extra signature bits.

As discussed earlier, the Bellare-Rogaway "PSS" and "PRab" proposals use a different technique to eliminate the space consumption of a long r.

By the way, here's something that's been bugging me. What most people now call the "RSA signature system" is a combination of three indispensable ideas:

(1) using roots modulo n as signatures;
(2) for security, hashing messages;
(3) for efficiency, using a small exponent.

Other ideas are certainly helpful, as explained above; but these are the three core ideas that produce a useful signature system of continuing interest today.

Rivest, Shamir, and Adleman published the first idea in [11] and [23]; I see no evidence that they came up with the second and third ideas. Rabin said in [21, page 156] that he had come up with the first idea independently; he then published the second and third ideas in [22].

Why, in light of this history, should these signature systems be credited to RSA and not Rabin? Even if we give RSA all the credit for the first idea—we do, after all, want to encourage quick publication—we should still be giving Rabin credit for the second and third ideas.

Most people do seem to suddenly start saying "Rabin signatures" when a fourth idea, exponent 2, is tossed into the mix. I see no reason to use any other exponent. Perhaps 2 will eventually become the most popular exponent, and, as a side effect, Rabin will receive more of the recognition that he deserves.

3. Standard signatures

This section specifies the signature system analyzed in this paper. In particular, it defines standard signatures and explains how to generate standard signatures.

This section does not present algorithms for key generation, key compression, signature verification, or signature compression. These algorithms, while important to implementors, are not relevant to the security analysis in this paper.

Parameters; key generation. The users of the system share four parameters:

- an integer $B \geq 0$ (for example, 4);
- an integer $K \geq 10$ (for example, 1536);
- a function H that maps strings of the form $(0, \dots)$ to $\{1, 2, \dots, 2^K\}$ and maps strings of the form $(1, \dots)$ to B-bit strings; and
- a distribution D (for example, the uniform distribution) of pairs of prime numbers (p, q) with $p \bmod 8 = 3$, $q \bmod 8 = 7$, and $2^K < pq < 2^{K+1}$.

B, K, D are fixed throughout this paper. H is not fixed; Section 5 averages over all functions H.

Key generation works as follows. The signer generates a uniform random 256-bit string z, and a random pair (p, q) from the distribution D. The signer's public key is the product pq. The signer's secret key is (p, q, z). It is also helpful for the signer to precompute the secrets $2^{(3p-5)/4} \bmod p$, $2^{(3q-5)/4} \bmod q$, and $q^{p-2} \bmod p$.

Many authors assume that the distribution D is uniform. In the real world, however, implementors often choose non-uniform distributions D to save time in key generation. This paper considers arbitrary distributions of pairs (p, q), and thus arbitrary distributions of public keys pq; it converts generic algorithms to forge signatures under those public keys into generic algorithms to factor those public keys.

Many authors specify a larger range of $H(0, \dots)$, namely $\{0, 1, \dots, pq - 1\}$, for the sake of convenience in averaging over all functions H. Of course, for H to actually produce a key-dependent range of output, H must be changed from a system parameter into a part of the public key. In this paper I make the more realistic assumption that H is independent of the key.

Messages; signature verification. A **message** is a string. An **H-signature** of a message m under a public key pq is a vector (e, f, r, s) such that $e \in \{1, -1\}$; $f \in \{1, 2\}$; r is a B-bit string; and s is an integer satisfying $fs^2 \equiv eH(0, r, m) \pmod{pq}$.

Signers are required to generate s in $\{0, 1, \dots, 2^K - 1\}$; in fact, in the smaller range $\{0, 1, \dots, (pq - 1)/2\}$. Thus s can be encoded in K bits, and a signature can be encoded in $K + B + 2$ bits. Verifiers are permitted to enforce these range requirements on s. However, this paper's security analysis does not require any such restriction.

Signatures can be expanded to accelerate verification, or compressed at only a slight cost in verification time. These operations are public, so they have no effect on security.

Standard signatures; signature generation. The **standard H-signature** of a message m under a secret key (p, q, z) is the unique vector (e, f, r, s) such that

- $r = H(1, z, m)$;
- $fs^2 \equiv eh \pmod{pq}$ where $h = H(0, r, m)$;
- e is 1 if h is a square modulo q, otherwise -1;
- f is 1 if eh is a square modulo p, otherwise 2;
- $s \in \{0, 1, \ldots, (pq - 1)/2\}$; and
- $\{s, pq - s\}$ contains a square modulo pq.

Note that the standard signature of m under (p, q, z) is a signature of m under pq. The existence of the standard signature is proven constructively; see Theorem 3.2. The uniqueness of the standard signature follows from the fact that, modulo pq, each square has a unique square root that is itself a square.

Algorithm 3.1. To print the standard signature of m under (p, q, z):

1. Compute $r \leftarrow H(1, z, m)$.
2. Compute $h \leftarrow H(0, r, m)$.
3. Compute $u \leftarrow h^{(q+1)/4} \bmod q$.
4. If $u^2 \equiv h \pmod{q}$, set $e \leftarrow 1$; otherwise set $e \leftarrow -1$.
5. Compute $v \leftarrow (eh)^{(p+1)/4} \bmod p$.
6. If $v^2 \equiv eh \pmod{p}$, set $f \leftarrow 1$; otherwise set $f \leftarrow 2$.
7. Compute $w \leftarrow f^{(3q-5)/4} u \bmod q$ and $x \leftarrow f^{(3p-5)/4} v \bmod p$.
8. Compute $y \leftarrow w + q(q^{p-2}(x - w) \bmod p)$.
9. If $2y < pq$, set $s \leftarrow y$; otherwise set $s \leftarrow pq - y$.
10. Print (e, f, r, s).

The signer can precompute $2^{(3q-5)/4} \bmod q$ and $2^{(3p-5)/4} \bmod p$ for Step 7, and $q^{p-2} \bmod p$ for Step 8, so Steps 1, 2, 3, and 5 take most of the time in Algorithm 3.1. Note that the algorithm does not require any Euclid-type computations of Jacobi symbols; it determines the squareness of h and eh much more efficiently by squaring u and v. I posted this idea on `sci.crypt` in October 2000.

Signers who worry about faults in their computations can double-check that y is a square modulo pq and that $fs^2 \equiv eh \pmod{pq}$. Both checks are very fast, and as above can be done without Euclid-type computations.

Theorem 3.2. *Algorithm 3.1 prints the standard signature of m under (p, q, z).*

Proof. If h is a square modulo q, say $h \equiv a^2 \pmod{q}$, then $u^2 \equiv h^{(q+1)/2} \equiv a^{q+1} \equiv a^2 \equiv h$, so $e = 1$ as claimed. Otherwise $u^2 \not\equiv h$ so $e = -1$ as claimed; but $u^4 \equiv h^{q+1} \equiv h^2$, so $u^2 \equiv -h$. Either way, $u^2 \equiv eh \pmod{q}$, so $f^{3(q-1)/2} u^2 \equiv eh \pmod{q}$; recall that $2^{(q-1)/2} \equiv 1 \pmod{q}$.

Similarly, if eh is a square modulo p, then $v^2 \equiv eh$ as above, so $f = 1$ as claimed. Otherwise $v^2 \not\equiv eh$ so $f = 2$ as claimed; also, $v^2 \equiv -eh$ as above. Either way, $f^{3(p-1)/2} v^2 \equiv eh \pmod{p}$; recall that $2^{(p-1)/2} \equiv -1 \pmod{p}$.

Next $y \equiv w \pmod q$, and $s \equiv \pm y$, so $fs^2 \equiv fy^2 \equiv fw^2 \equiv ff^{(3q-5)/2}u^2 = f^{3(q-1)/2}u^2 \equiv eh \pmod q$. Similarly, $y \equiv w + q^{p-1}(x-w) \equiv x \pmod p$, and $s \equiv \pm y$, so $fs^2 \equiv fy^2 \equiv fx^2 \equiv ff^{(3p-5)/2}v^2 = f^{3(p-1)/2}v^2 \equiv eh \pmod p$. Thus $fs^2 \equiv eh \pmod{pq}$.

Furthermore, $y \equiv f^{(3q-5)/4}h^{(q+1)/4} \pmod q$, and $y \equiv f^{(3p-5)/4}(eh)^{(p+1)/4} \equiv f^{(3p-5)/4}(eh)^{(p-3)/4}eh \equiv f^{(3p-5)/4}(eh)^{(p-3)/4}f^{3(p-1)/2}v^2 \equiv f^{(9p-11)/4}(eh)^{(p-3)/4}v^2 \pmod p$. All the exponents here are even, so y is a square modulo both p and q, hence modulo pq; so $\{s, pq-s\} = \{y, pq-y\}$ contains a square modulo pq.

Finally, $0 \le w \le q-1$ and $0 \le (q^{p-2}(x-w) \bmod p) \le p-1$, so $0 \le y \le q-1+q(p-1) = pq-1$. If $y \le (pq-1)/2$ then $s = y \in \{0, 1, \ldots, (pq-1)/2\}$ as claimed; otherwise $y \ge (pq+1)/2$ so $s = pq - y \in \{0, 1, \ldots, (pq-1)/2\}$ as claimed. □

4. Squares

Consider vectors (e, f, s, h) obtained as follows: a signer generates a uniform random number $h \in \{1, 2, \ldots, 2^K\}$, and then uses his secrets p and q to generate (e, f, s) from h as in Algorithm 3.1. This section explains how to generate vectors (e, f, s, h) with the same distribution, using only the public key pq. The conclusion in Section 5 will be that an attacker doesn't learn anything from such vectors.

How do we find a square root s of a random number h modulo pq without knowing p and q? Answer: Generate s first, with the right distribution, and then square it to obtain h.

How do we also ensure that $\pm s$ is a square, as the signer would have done? Answer: Generate the square root of $\pm s$ first, with the right distribution. A different answer is to check the Jacobi symbol of s; but that isn't as fast as squaring.

The rest of this section goes through the details of properly handling e, f, K, etc. and generating exactly the right distribution. Algorithm 4.1, with $c = 1$, prints the advertised vectors (e, f, s, h).

Algorithm 4.1. Given a public key n and an integer c coprime to n, to print (e, f, s, h):

1. Generate a uniform random $x \in \{0, 1, \ldots, 2^{K+1} - 1\}$.
2. Generate a uniform random $e' \in \{-1, 1\}$.
3. Generate a uniform random $f' \in \{1, 2\}$.
4. If $x \ge n$, go back to Step 1.
5. Set $g \leftarrow \gcd\{x, n\}$.
6. If $g = n$ or $g \bmod 8 = 7$, set $e \leftarrow 1$; otherwise set $e \leftarrow e'$.
7. If $g = n$ or $g \bmod 8 = 3$, set $f \leftarrow 1$; otherwise set $f \leftarrow f'$.
8. Set $y \leftarrow x^2 \bmod n$.
9. If $2y < n$, set $s \leftarrow y$; otherwise set $s \leftarrow n - y$.
10. Set $h \leftarrow ef(cs)^2 \bmod n$.
11. If $h \le 0$ or $h > 2^K$, go back to Step 1.
12. Print (e, f, s, h).

Theorem 4.2. *Let $n = pq$ be a public key, and let c be an integer coprime to n. For each $h \in \{1, 2, \ldots, 2^K\}$, there are exactly 4 choices of (x, e', f') in Steps 1–3 of Algorithm 4.1 that produce h in Step 12 without going back to Step 1. Furthermore, if Algorithm 4.1 prints (e, f, s, h), then e is 1 if h is a square modulo q, otherwise -1; and f is 1 if eh is a square modulo p, otherwise 2.*

Proof. Recall that each nonzero square modulo p has exactly two fourth roots modulo p; I will write those roots as $\pm\sqrt[4]{\ }$. The same comment applies to q.

Assume that Algorithm 4.1 prints (e, f, s, h). Then $h \equiv efc^2s^2 \equiv efc^2y^2 \equiv efc^2x^4 \pmod{n}$. By construction $e \in \{1, -1\}$ and $f \in \{1, 2\}$, so ef is coprime to n, as is c. There are three possibilities for the quadratic character of h modulo q:

- $h \bmod q = 0$. Then $x \bmod q = 0$, so g is either n or q, so $e = 1$ in Step 6; e' could be either 1 or -1.
- h is a nonzero square modulo q. Then $-h$ is not a square, so e cannot be -1, so $e = 1$, so $x \equiv \pm\sqrt[4]{h/fc^2} \pmod{q}$. Furthermore, $g \ne n$ and $g \ne q$, so $e' = e$.
- h is a nonsquare modulo q. Then e cannot be 1, so $e = -1$, so $x \equiv \pm\sqrt[4]{-h/fc^2} \pmod{q}$. Furthermore, $e' = e$ as above.

Similarly, there are three possibilities for the quadratic character of eh modulo p:

- $eh \bmod p = 0$. Then $x \bmod p = 0$, so g is either n or p, so $f = 1$ in Step 7; f' could be either 1 or 2.
- eh is a nonzero square modulo p. Then $eh/2$ is not a square, so f cannot be 2, so $f = 1$, so $x \equiv \pm\sqrt[4]{eh/c^2} \pmod{p}$. Furthermore, $f' = f$ as above.
- eh is a nonsquare modulo p. Then f cannot be 1, so $f = 2$, so $x \equiv \pm\sqrt[4]{eh/2c^2} \pmod{p}$. Furthermore, $f' = f$ as above.

In every case, h determines (e, f); there are exactly 2 possibilities for $(x \bmod q, e')$; and there are exactly 2 possibilities for $(x \bmod p, f')$. Thus there are exactly 4 possibilities for $(x \bmod n, e', f')$. Step 4 restricts attention to $x \in \{0, 1, \ldots, n-1\}$, so there are exactly 4 possibilities for (x, e', f').

Conversely, if (x, e', f') is one of these 4 possibilities, then Algorithm 4.1 does not stop in Step 4; it produces the e and f noted above; it produces h in Step 10; and it does not stop in Step 11. $\square$

Theorem 4.3. *Let $n = pq$ be a public key, and let c be an integer coprime to n. Given n and c, Algorithm 4.1 prints a random vector (e, f, s, h) such that h is a uniform random element of $\{1, 2, \ldots, 2^K\}$; $f(cs)^2 \equiv eh \pmod{n}$; e is 1 if h is a square modulo q, otherwise -1; f is 1 if eh is a square modulo p, otherwise 2; $s \in \{0, 1, \ldots, (n-1)/2\}$; and $\{s, n-s\}$ contains a square modulo n.*

Proof. By Step 8, y is a square modulo n, and $0 \le y \le n-1$. By Step 9, $s \in \{0, 1, \ldots, (n-1)/2\}$; also, $\{s, n-s\} = \{y, n-y\}$, so $\{s, n-s\}$ contains a square modulo n. By Step 10, $eh \equiv e^2f(cs)^2 = f(cs)^2 \pmod{n}$. By Step 11, $1 \le h \le 2^K$. Theorem 4.2 guarantees the choice of e, the choice of f, and the uniformity of h. $\square$

Notes on speed. Algorithm 4.1 is applied in two ways in Section 5. In the first application, c is taken as 1, so the multiplication by c can be omitted from Step 10. In the second application, there is no need for s to be a square or in a limited range, so Steps 8 and 9 can be replaced by "Set $s \leftarrow x$."

The cost of Step 5 can be amortized over many invocations of Algorithm 4.1, as in Pollard's ρ method of factorization: multiply all the x's together and do a single gcd to see whether any of them has a factor in common with n, recovering appropriately if so. Even better, one can skip all these gcd computations; the chance of $g \ne 1$ is negligible, so setting $(e, f) \leftarrow (e', f')$ does not noticeably affect the distribution of (e, f, s, h). However, generating *exactly* the right distribution simplifies the proof in Section 5.

Out of the 2^{K+3} choices of (x, e', f'), exactly $4n$ survive Step 4, and exactly 2^{K+2} also survive Step 11, by Theorem 4.2. Thus Algorithm 4.1 loops exactly twice on average. One can reduce the number of loops to about $n/2^K$ by generating $x \in \{0, 1, \ldots, 2^{K+10} - 1\}$, starting over if $\lfloor x/n \rfloor = \lfloor 2^{K+10}/n \rfloor$, and then replacing x with $x \bmod n$. The remaining gap between $n/2^K$ and 1 accounts for the limited range of h. Of course, the gap is small if n is close to 2^K.

5. Security

This section explains how to convert an attack A into a factorization algorithm F; compares the success probability of F to the *generic* success probability of A; compares the speed of F to the speed of A; and discusses some variants of the proof.

Attacks; generic success probability. An attacker is faced with the following situation. The signer generates a public key n and gives it to the attacker. The attacker provides a message to the signer, and receives the standard signature of the message; the attacker provides another message to the signer, and receives the standard signature of that message; and so on. Eventually the attacker prints a forgery (m, e', f', r, s'). This forgery is **successful** if (e', f', r, s') is a signature of m under n and m is different from any of the messages provided to the signer.

Of course, if the attacker actually has this much power, the attacker doesn't need to forge signatures: he can simply provide a message of his choice to the signer. Any signer in the real world will restrict the messages signed, and thus restrict the set of possible attacks. But the security proof does not depend on any such restrictions.

More formally, an **attack** is an algorithm given access to two oracles, a **hashing oracle** and a **signing oracle**. The algorithm receives a random public key n as input and prints a forgery as output. The **success probability** of the algorithm against a function H is the probability that the algorithm prints a successful H-forgery under n, when it uses H as its hashing oracle and "standard H-signature under n" as its signing oracle. The **generic success probability** of the algorithm is its success probability against a uniform random function H; in other words, the average of its success probabilities against all functions H.

Consider, for example, an algorithm that factors n and then computes a signature of any desired message with Algorithm 3.1, using the hashing oracle to evaluate H. This algorithm has success probability 1 against all functions H, and thus generic success probability 1, but it is extremely slow with our current factorization algorithms when K is large.

As another example, fix $K = 1536$, and consider an algorithm that looks for forgeries $(m, 1, 1, r, s)$ by feeding 2^{50} different pairs (r, m) to H and checking whether $H(0, r, m)$ is an integer square s^2 with $1 \le s \le 2^{768}$. This algorithm has generic success probability 2^{-718}, but it has success probability 1 against a function H that maps $(0, r, m)$ to $\mathrm{MD5}(0, r, m)^2$. As this example illustrates, a small generic success probability does not guarantee a small success probability for any particular function H.

Converting an attack A into a factorization algorithm F. Fix an attack A. The following algorithm F, given n as input, tries to use A to factor n.

F begins by choosing a uniform random integer c such that $c \in \{0, 1, \ldots, n-1\}$ and c is coprime to n. F also chooses a uniform random 256-bit string z.

F builds random functions Q and H as follows. For each string x, define $H(1, x)$ as a uniform random B-bit string. For each pair (r, m) where r is a B-bit string and m is a string, define $Q(r, m)$ as a random vector (e, f, s, h) produced by Algorithm 4.1 with input $(n, 1)$ if $r = H(1, z, m)$ or with input (n, c) if $r \neq H(1, z, m)$; and define $H(0, r, m)$ as h. Note that the distribution of H is uniform by Theorem 4.3.

Of course, F does not build the entire function H at once. It generates the values of H and Q only upon demand, saving those values in a table for future reference.

F now runs A with this function H. The function Q allows F to compute the standard signature of any requested message m as (e, f, r, s) where $r = H(1, z, m)$ and $(e, f, s, h) = Q(r, m)$. Indeed, (e, f, s, h) is an output of Algorithm 4.1 with input $(n, 1)$, by definition of Q; $h = H(0, r, m)$, by definition of H; e and f are chosen properly, s is in the right range, $\{s, n-s\}$ contains a square modulo n, and $fs^2 \equiv eh \pmod{n}$, by Theorem 4.3; so (e, f, r, s) is the standard signature of m as desired.

Eventually A produces a forgery (m, e', f', r, s'). If $r \neq H(1, z, m)$ then F looks up $(e, f, s, h) = Q(r, m)$ and computes both $\gcd\{n, s'\}$ and $\gcd\{n, s' - cs\}$, hoping to obtain a nontrivial factor of n. That's it.

Effectiveness of F. Because F runs A with a uniform random function H, the probability that A generates a successful forgery inside F is exactly the generic success probability of A. Define ϵ as this probability, and assume that A provides at most α queries of the form $(1, \ldots)$ to the hashing oracle. What is the probability that F successfully uses A's output to factor n?

All bets are off if A's queries to the hashing oracle included $(1, z, m)$ for some m. This happened with probability at most $2^{-256}\alpha$. Assume that it did not, in fact, happen, and that A's forgery (m, e', f', r, s') is successful; this happens with probability at least $\epsilon - 2^{-256}\alpha$.

By definition of "successful forgery," A never provided m as a query to the signing oracle. Thus F never evaluated $H(1, z, m)$ while A was running; so the value of $H(1, z, m)$ is independent of A's output. F now checks whether $r = H(1, z, m)$; this happens with conditional probability at most $1/2^B$.

Assume that, in fact, $r \neq H(1, z, m)$. F now looks up $(e, f, s, h) = Q(r, m)$. This is, by definition of Q, an output of Algorithm 4.1 with input (n, c); by Theorem 4.3, $fc^2s^2 \equiv eh \pmod{n}$. On the other hand, $f'(s')^2 \equiv e'h \pmod{n}$, by definition of "successful forgery." Hence $efc^2s^2 \equiv e'f'(s')^2 \pmod{n}$.

If s' has a factor in common with n then $\gcd\{n, s'\}$ is a nontrivial factor of n: recall that $h \neq 0$, so s' cannot be divisible by n.

If, on the other hand, s' is coprime to n, then $e'f'/ef \equiv c^2s^2/(s')^2 \pmod{n}$; but none of $\{-1, 2, -2, 1/2, -1/2\}$ are squares modulo n, so ef must equal $e'f'$. Thus $c^2s^2 \equiv (s')^2 \pmod{n}$; i.e., c is a square root of $(s'/s)^2$ modulo n. The choice of square root is independent of A's output: Algorithm 4.1 depends solely on $c^2 \bmod n$. Thus the conditional probability is $1/2$ that c is neither s'/s nor $-s'/s$, and hence that $\gcd\{n, s' - cs\}$ is a nontrivial factor of n.

To summarize: F factors n with probability at least $(\epsilon - 2^{-256}\alpha)(1 - 1/2^B)/2$.

Efficiency of F. There are several major costs in this factorization algorithm:

- Generating B random bits for $H(1, x)$. This happens at most once for each of A's oracle calls, plus once at the end of F.
- Algorithm 4.1. Each call involves generating, on average, $2K + 6$ random bits, and performing 8 multiplications mod n; see Section 4. This happens at most once for each of A's oracle calls, plus once at the end of F. For comparison, when A is used to attack the signature system, each call to the hashing oracle means evaluating H, and each call to the signing oracle means waiting for the signer to run Algorithm 3.1.
- Maintaining and consulting tables of values of H and Q. Data structures such as B-trees are reasonably fast in simple models of computation that do not charge for space, but they could easily dominate the cost of F in more realistic models of computation. An alternative is to scrap the tables in favor of a fast, conjecturally strong pseudorandom number generator; [12] credits this idea to Levin. Of course, it's conceivable that F will end up breaking the generator rather than factoring n—there are no known generators with *good* reductions to factoring.
- A's internal computations. This is the bottleneck for some attacks A, but there is no reason to believe that it is always the bottleneck.

For example, consider again the (low-probability) attack A that looks for integer squares among many values of $H(0, r, m)$. The corresponding (low-probability) factorization algorithm will be an order of magnitude slower than signature forgery if Algorithm 4.1 is an order of magnitude slower than H.

Variants. The crucial objects in the construction of F are

- a set U of hash values, namely $\{1, 2, \dots, 2^K\}$;
- a set S_1 of signature vectors, namely the set of vectors $(e, f, s) \in \{-1, 1\} \times \{1, 2\} \times \mathbf{Z}$ such that $efs^2 \bmod n \in U$;
- a surjective verifying function $v_1 : S_1 \to U$, namely the function $(e, f, s) \mapsto efs^2 \bmod n$;
- a subset $T_1 \subseteq S_1$ such that v_1 is a bijection from T_1 to U, namely the set of standard vectors, defined in the obvious way;
- an algorithm A_1, namely Algorithm 4.1, to generate a uniform random element of T_1; and
- analogous sets S_2, v_2, T_2, A_2, defined using $ef(cs)^2$ instead of efs^2.

With probability at least $(\epsilon - 2^{-256}\alpha)(1 - 1/2^B)$, the algorithm F obtains a **claw**: a pair $(x_1, x_2) \in S_1 \times S_2$ such that $v_1(x_1) = v_2(x_2)$. It then uses this claw, namely $((e', f', s'), (e, f, s))$, to factor n with probability at least $1/2$.

In the special case $S_1 = T_1 = S_2 = T_2 = U$, generating a uniform random element of T_1 or T_2 means simply generating a uniform random element of U. This is the "claw-free permutation pair" setting discussed in [12] and [13]. In contrast, in the Rabin-Williams setting, the set of standard signatures T_1 is smaller and more complicated than the set of signatures S_1.

One could improve $(\epsilon - 2^{-256}\alpha)(1 - 1/2^B)/2$ to $(\epsilon - 2^{-256}\alpha)(1 - 1/2^B)$ by changing S_1 and S_2, imposing the additional condition that the Jacobi symbol of s modulo n be 1. Then s'/s and $-s'/s$ both have Jacobi symbol 1; so the choice $c = 2$, with Jacobi symbol -1, guarantees a factorization from any successful forgery. But I wouldn't use this variant in practice: the Jacobi-symbol computation would make signature verification much slower.

Coron in [8] suggested a way to handle a smaller set U, using an idea of Vallée for finding small squares modulo n. The speed of Coron's construction hasn't been quantified. My impression is that the construction is substantially slower than Algorithm 4.1, so it will need a larger value of K, countering any possible savings from faster functions H.

References

[1] Victoria Ashby (editor), *First ACM conference on computer and communications security*, Association for Computing Machinery, New York, 1993.

[2] Mihir Bellare (editor), *Advances in cryptology—CRYPTO 2000: proceedings of the 20th Annual International Cryptology Conference held in Santa Barbara, CA, August 20–24, 2000*, Lecture Notes in Computer Science, 1880, Springer-Verlag, Berlin, 2000. ISBN 3–540–67907–3. MR 2002c:94002.

[3] Mihir Bellare, Phillip Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, in [1] (1993), 62–73.

[4] Mihir Bellare, Phillip Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin*, in [19] (1996), 399–416; see also newer version in [5].

[5] Mihir Bellare, Phillip Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin* (1996); see also older version in [4]. Available from `http://www-cse.ucsd.edu/~mihir/papers/exactsigs.html`.

[6] Jean-Sébastien Coron, *On the exact security of Full Domain Hash*, in [2] (2000), 229–235. MR 2002e:94109. Available from `http://www.eleves.ens.fr/home/coron/publications/publications.html`.

[7] Jean-Sébastien Coron, *Optimal security proofs for PSS and other signature schemes*, in [14] (2002), 272–287. Available from `http://www.eleves.ens.fr/home/coron/publications/publications.html`.

[8] Jean-Sébastien Coron, *Security proof for partial-domain hash signature schemes*, in [26] (2002), 613–626. Available from `http://www.gemplus.com/smart/r_d/publications/`.

[9] Richard A. DeMillo, David P. Dobkin, Anita K. Jones, Richard J. Lipton (editors), *Foundations of secure computation*, Academic Press, New York, 1978. ISBN 0–12–210350–5.

[10] Amos Fiat, Adi Shamir, *How to prove yourself: practical solutions to identification and signature problems*, in [20] (1987), 186–194. MR 88m:94023.

[11] Martin Gardner, *A new kind of cipher that would take millions of years to break*, Scientific American (1977), 120–124.

[12] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal on Computing **17** (1988), 281–308. ISSN 0097–5397. MR 89e:94009. Available from `http://theory.lcs.mit.edu/~rivest/publications.html`.

[13] Jonathan Katz, Nan Wang, *Efficiency improvements for signature schemes with tight security reductions* (2003). Available from `http://www.cs.umd.edu/~jkatz/papers.html`.

[14] Lars Knudsen (editor), *Advances in cryptology—EUROCRYPT 2002: proceedings of the 21st International Annual Conference on the Theory and Applications of Cryptographic Techniques held in Amsterdam, April 28–May 2, 2002*, Lecture Notes in Computer Science, 2332, Springer-Verlag, Berlin, 2002. ISBN 3–540–43553–0.

[15] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 1st edition, 1st printing, Addison-Wesley, Reading, 1969; see also newer version in [16]. MR 44:3531.

[16] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 1st edition, 2nd printing, Addison-Wesley, Reading, 1971; see also older version in [15]; see also newer version in [17]. MR 44:3531.

[17] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, Reading, 1981; see also older version in [16]; see also newer version in [18]. ISBN 0–201–03822–6. MR 83i:68003.

[18] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 3rd edition, Addison-Wesley, Reading, 1997; see also older version in [17]. ISBN 0–201–89684–2.

[19] Ueli M. Maurer (editor), *Advances in cryptology—EUROCRYPT '96: Proceedings of the Fifteenth International Conference on the Theory and Application of Cryptographic Techniques held in Saragossa, May 12–16, 1996*, Lecture Notes in Computer Science, 1070, Springer-Verlag, Berlin, 1996. ISBN 3–540–61186–X. MR 97g:94002.

[20] Andrew M. Odlyzko (editor), *Advances in cryptology—CRYPTO '86: proceedings of the conference on the theory and applications of cryptographic techniques held at the University of California, Santa Barbara, Calif., August 11–15, 1986*, Lecture Notes in Computer Science, 263, Springer-Verlag, Berlin, 1987. ISBN 3–540–18047–8. MR 88h:94004.

[21] Michael O. Rabin, *Digitalized signatures*, in [9] (1978), 155–168. Available from `http://cr.yp.to/bib/entries.html#1978/rabin`.

[22] Michael O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report 212, MIT Laboratory for Computer Science, 1979. Available from `http://ncstrl.mit.edu/Dienst/UI/2.0/Describe/ncstrl.mit_lcs/MIT/LCS/TR-212`.

[23] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), 120–126. ISSN 0001–0782. Available from `http://cr.yp.to/bib/entries.html#1978/rivest`.

[24] Douglas R. Stinson, *Cryptography: theory and practice*, CRC Press, Boca Raton, Florida, 1995. ISBN 0–8493–8521–0. MR 96k:94015.

[25] Hugh C. Williams, *A modification of the RSA public-key encryption procedure*, IEEE Transactions on Information Theory **26** (1980), 726–729. ISSN 0018–9448. Available from `http://cr.yp.to/bib/entries.html#1980/williams`.

[26] Moti Yung (editor), *Advances in cryptology—CRYPTO 2002: 22nd annual international cryptology conference, Santa Barbara, California, USA, August 2002, proceedings*, Lecture Notes in Computer Science, 2442, Springer-Verlag, Berlin, 2002. ISBN 3–540–44050–X.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@cr.yp.to`